Financiometrics Inc.
Tel : (925) 254-9338
Fax: (925) 254-2932

LSO-MAX™

Long-Short Optimization System

The top of the line long-short optimizer.

Features

LSO-MAX is an exceptionally fast long-short portfolio optimizer based on the active set method for quadratic programming. It allows long positions to migrate to short positions and vice versa, and it takes advantage of the structure of the long-short optimization problem to optimize large portfolios 1,000 times faster than a general purpose quadratic programming algorithm. It has special features that can be used to:

- Set a gross leverage constraint.
- Set transactions costs that vary individually for each asset based on transaction size.
- Set linear equality and inequality constraints for portfolio attributes and asset-group weights.
- Set quadratic penalty functions around targeted portfolio attribute values and asset-group weights.
- Set a turnover constraint.
- Set upper and lower bounds on asset weights individually for each asset.
- Set quadratic penalty functions around targeted asset weights individually for each asset.
- Include risk-free assets.
- Interface with an in-house factor model, or a factor model supplied by a vendor.
- Maximize expected active return for a given standard deviation of active return, where active return is measured relative to a benchmark or normal portfolio.
- Minimize standard deviation of active return for a given expected

active return, where active return is measured relative to a benchmark or normal portfolio.
- Maximize the mean-variance utility function for active return.
- Maximize expected total return for a given standard deviation of total return.
- Minimize standard deviation of total return for a given expected total return.
- Maximize the mean-variance utility function for total return.
- Maximize the Sharpe ratio for a given threshold return.
- Maximize the threshold return for a given Sharpe ratio.

Limits

None

Operating Environments

The LSO-MAX optimization software is available as a subroutine library. We support the following combinations of computer platforms and languages:

- Dynamic link library (DLL) for Windows, for use with C/C++, Visual Basic, or Fortran.
- Subroutine library for Linux, Sun Solaris and Windows.
- C/C++ and Fortran source code is available for all computer platforms.

Internet and Intranet Systems

The C/C++ subroutine library is thread-safe. You can encapsulate it with a C/C++ wrapper to build Internet and Intranet systems for Linux, Sun Solaris, and Windows platforms.

System Support

We provide support by e-mail and telephone during business hours Pacific Time, U.S.A.

Consulting

We offer consulting services for building in-house systems that incorporate LSO-MAX.

Sign in

Google

Web Images Video News Maps more »

sun solaris optimizer threshold Search Advanced Search Preferences

Web Results **1 - 10** of about **32,200** for **sun solaris optimizer threshold** with **Safesearch on**. (**0.22** seconds)

[PDF] Microsoft PowerPoint - EPIC-6-Keynote-PSI.ppt
File Format: PDF/Adobe Acrobat - View as HTML
Sun - Solaris. Linux. Windows. z/OS. IBM AIX. HP-UX. **Sun - Solaris** ... Invoke **optimization** only after a certain instruction re-use **threshold** is. reached ...
rogue.colorado.edu/EPIC6/EPIC-6-Keynote-PSI.pdf - Similar pages

Help - Eclipse Platform
Installation Guide for **Sun Solaris** ... Operator name alignment for the Abstract Plan and the **optimizer** criteria · Extending the **optimizer** criteria set ...
infocenter.sybase.com/help/index.jsp - 977k - May 10, 2007 - Cached - Similar pages

[PDF] Websense Enterprise Bandwidth **Optimizer**
File Format: PDF/Adobe Acrobat - View as HTML
bandwidth **threshold** than the sales department, while everyone is ... Pentium III or **Sun** Ultra 10 processor or greater with at least 512 MB ...
www.sd.co.za/downloads_vendors/websense/Websense_Bandwidth**Optimizer**.pdf - Similar pages

Financiometrics Inc. - QOS-MAX
QOS-MAX is an exceptionally fast portfolio **optimizer** based on the active set method for quadratic ... Subroutine library for Linux, **Sun Solaris** and Windows. ...
www.financiometrics.com/qos-max.htm - 11k - Cached - Similar pages

> Financiometrics Inc. - QOS-MAX
> LSO-MAX is an exceptionally fast long-short portfolio **optimizer** based on the active set method ... Subroutine library for Linux, **Sun Solaris** and Windows. ...
> www.financiometrics.com/lso-max.htm - 9k - Cached - Similar pages

Book Detail Information
High Availability on **Sun Solaris**. High Availability with VERITAS Cluster Server. ... The SQL Compiler and the Query **Optimizer**. Compensation. ...
opamp.com/cf/details.cfm?ISBN=0130463884 - 20k - Cached - Similar pages

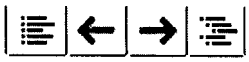
[PPT] More Data...More Easily
File Format: Microsoft Powerpoint - View as HTML
Optimize IT investments given more choice in data access; Integrate data with better ... AIX v5.2; **Sun Solaris** 9; HP-UX IPF 11i; Windows32 on Server 2003 ...
www.iiug.org/resources/webcasts/postcast/19nov03_RedBrick.ppt - Similar pages

HP OpenView Storage Area Manager (SAM) 3.2 - World Wide QuickSpecs
Storage **Optimizer** relies on HBA SNIA libraries to gather performance information for hosts. ... **Sun Solaris** 8.0, 9.0; Red Hat Linux Advanced Server 2.1 ...
h18000.www1.hp.com/products/quickspecs/11460_div/11460_div.HTML - 68k - Cached - Similar pages

Release Bulletin Adaptive Server Enterprise 12.5.3a for **Sun** ...
Adaptive Server is supported on the following **Sun Solaris** platform: ... and then applied the **threshold** procedure to one or more database segments, ...
www.sybase.com/detail?id=1037768 - 190k - Cached - Similar pages

Q10.3.1: SYBASE Technical News, Volume 5, Number 1, Feb 1996
A fix for an **optimizer** bug, 13495 (also listed as 17230), changed the behavior of the **optimizer**. ... **Sun Solaris** 2.x, 4157. Digital OpenVMS Alpha 1.5, 4158 ...
www.isug.com/Sybase_FAQ/ASE/Section10/3/Q10.3.1.html - 34k - Cached - Similar pages

Result Page: 1 2 3 4 5 6 7 8 9 10 **Next**



CHAPTER 3

SPARC Optimization and Parallel Processing

This chapter describes how to use compiler and linking options to optimize applications for:

- Specific SPARC® instruction set architectures
- 64-bit enabled Solaris operating environment
- Parallel processing on SPARC platforms

TABLE 3-1 shows a comparison of the 32-bit and 64-bit operating environments. These items are described in greater detail in the following sections.

TABLE 3-1 Comparison of 32-bit and 64-bit Operating Environ

	32-bit (ILP 32)	64-bit (LP64)
`-xarch`	`v8`, `v8plusa`, `v8plusb`	`v9`, `v9a`, `v9b`
Fortran Integers	`INTEGER`, `INTEGER*4`	`INTEGER*8`
C Integers	`int`	`long`
Floating-point	`S/D/C/Z`	`S/D/C/Z`
API	Names of routines	Names of routines with `_64` suffix

Using Sun Performance Library on SPARC Platforms

The Sun Performance Library was compiled using the `f95` compiler provided with this release. The Sun Performance Library routines were compiled using `-dalign`, `-xparallel`, and `-xarch` set to `v8`, `v8plusa`, or `v9a`.

When linking the program, use `-dalign`, `-xlic_lib=sunperf`, and the same command line options that

were used when compiling. If `-dalign` cannot be used in the program, supply a trap 6 handler as described in Getting Started With Sun Performance Library. If compiling with a value of `-xarch` that is not one of `[v8|v8plusa|v9a]`, the compiler driver will select the closest match.

Sun Performance Library is linked into an application with the `-xlic_lib` switch rather than the `-l` switch that is used to link in other libraries, as shown here.

```
my_system% f95 -dalign my_file.f -xlic_lib=sunperf
```

Compiling for SPARC Platforms

Applications using Sun Performance Library can be optimized for specific SPARC instruction set architectures and for a 64-bit enabled Solaris operating environment. The optimization for each architecture is targeted at one implementation of that architecture and includes optimizations for other architectures when it does not degrade the performance of the primary target.

Compile with the most appropriate `-xarch=` option for best performance. At link time, use the same `-xarch=` option that was used at compile time to select the version of the Sun Performance Library optimi

---

Note - Using SPARC-specific optimization options increases application performance on the selected instruction set architecture, but limits code portability. When using these optimization options, the resulting code can be run only on systems using the specific SPARC chip from Sun Microsystems and, in some cases, a specific Solaris operating environment (32-bit or 64-bit Solaris 7, Solaris 8, or Solaris 9).

The SunOS™ command `isalist`(1) can be used to display a list of the native instruction sets executable on a particular platform. The names output by `isalist` are space-separated and are ordered in the sense of best performance.

For a detailed description of the different `-xarch` options, refer to the *Fortran User's Guide* or the *C User's Guide*.

Use the following command line options to compile for 32-bit addressing in a 32-bit enabled Solaris operating environment:

- **UltraSPARC I™ or UltraSPARC II™ systems.** Use `-xarch=v8plus` or `-xarch=v8plusa`.
- **UltraSPARC III™ systems.** Use `-xarch=v8plus` or `-xarch=v8plusb`.

Use the following command line options to compile for 64-bit addressing in a 64-bit enabled Solaris operating environment.

- **UltraSPARC I or UltraSPARC II systems.** Use `-xarch=v9` or `-xarch=v9a`.

- **UltraSPARC III systems.** Use `-xarch=v9` or `-xarch=v9b`.

Compiling Code for a 64-Bit Enabled Solaris Operating Environment

To compile code for a 64-bit enabled Solaris operating environment, use `-xarch=v9[a|b]` and convert all integer arguments to 64-bit arguments. 64-bit routines require the use of 64-bit integers.

Sun Performance Library provides 32-bit and 64-bit interfaces. To use the 64-bit interfaces:

- **Modify the Sun Performance Library routine name.** For C and Fortran 95 code, append `_64` to the names of Sun Performance Library routines (for example, `rfftf_64` or `CFFTB_64`). For Fortran 95 code with the `USE SUNPERF` statement, the `_64` suffix is not strictly required for specific interfaces, such as `DGEMM`. The `_64` suffix is still required for the generic interfaces, such as `GEMM`.

- **Promote integers to 64 bits.** Double precision variables and the real and imaginary parts of double complex variables are already 64 bits. Only the integers are promoted to 64 bits.

64-Bit Integer Arguments

These additional 64-bit-integer interfaces are available only in the v9, v9a, and v9b libraries. Codes compiled for 32-bit operating environments (`-xarch` set to `v8plusa` or `v8plusb`) can not call the 64-bit-integer interfaces.

To call the 64-bit-integer interfaces directly, append the suffix `_64` to the standard library name. For example, use `daxpy_64()` in place of `daxpy()`.

However, if calling the 64-bit integer interfaces indirectly, do not append `_64` to the name of the Sun Performance Library routine. Calls to the Sun Performance Library routine will access a 32-bit wrapper that promotes the 32-bit integers to 64-bit integers, calls the 64-bit routine, and then demotes the 64-bit integers to 32-bit integers.

For best performance, call the routine directly by appending `_64` to the routine name.

For C programs, use `long` instead of `int` arguments. The following code example shows calling the 64-bit integer interfaces directly.

```
#include <sunperf.h>

long n, incx, incy;

double alpha, *x, *y;

daxpy_64(n, alpha, x, incx, y, incy);
```

The following code example shows calling the 64-bit integer interfaces indirectly.

```
#include <sunperf.h>

int  n, incx, incy;

double alpha, *x, *y;

daxpy   (n, alpha, x, incx, y, incy);
```

For Fortran programs, use 64-bit integers for all integer arguments. The following methods can be used to convert integer arguments to 64-bits:

- To promote all default integers (integers declared without explicit byte sizes) and literal integer constants from 32 bits to 64 bits, compile with `-xtypemap=integer:64`.
- To promote specific integer declarations, change `INTEGER` or `INTEGER*4` to `INTEGER*8`.
- To promote integer literal constants, append `_8` to the constant.

Consider the following code example.

```
INTEGER*8 N

REAL*8 ALPHA, X(N), Y(N)


! _64 SUFFIX: N AND 1_8 ARE 64-BIT INTEGERS

CALL DAXPY_64(N,ALPHA,X,1_8,Y,1_8)
```

`INTEGER*8` arguments cannot be used in a 32-bit environment. Routines in the 32-bit libraries, `v8`, `v8plu`

When passing constants in Fortran 95 code that have not been compiled with `-xtypemap`, append `_8` to literal constants to effect the promotion. For example, when using Fortran 95, change `CALL DSCAL (20,5.26D0,X,1)` to `CALL DSCAL(20_8,5.26D0,X,1_8)`. This example assumes `USE SUNPERF` is incluc

The following code example shows calling `CAXPY` from Fortran 95 using 32-bit arguments.

```
      PROGRAM TEST

      COMPLEX ALPHA

      INTEGER INCX, INCY, N

      COMPLEX X(*), Y(*)
```

```
      CALL CAXPY(N, ALPHA, X, INCX, Y, INCY)
```

The following code example shows calling CAXPY from Fortran 95 (without the USE SUNPERF statement) using 64-bit arguments.

```
      PROGRAM TEST
      COMPLEX    ALPHA
      INTEGER*8 INCX, INCY, N
      COMPLEX    X(*), Y(*)


      CALL CAXPY_64(N, ALPHA, X, INCX, Y, INCY)
```

When using 64-bit arguments, the _64 must be appended to the routine name if the USE SUNPERF statement is not used.

The following Fortran 95 code example shows calling CAXPY using 64-bit arguments.

```
      PROGRAM TEST
      USE SUNPERF
      .
      .
      .
      COMPLEX    ALPHA
      INTEGER*8 INCX, INCY, N
      COMPLEX    X(*), Y(*)


      CALL CAXPY(N, ALPHA, X, INCX, Y, INCY)
```

In C routines, the size of long is 32 bits when compiling for V8 or V8plus and 64 bits when compiling for V9. The following code example shows calling the dgbcon routine using 32-bit arguments.

```
void dgbcon(char norm, int n, int nsub, int nsuper, double *da,
```

```
            int lda, int *ipivot, double danorm, double drcond,
            int *info)
```

The following code example shows calling the `dgbcon` routine using 64-bit arguments.

```
void dgbcon_64 (char norm, long n, long nsub, long nsuper,
                  double *da, long lda, long *ipivot, double danorm,
                double *drcond, long *info)
```

Parallel Processing on SPARC Platforms

To enable parallel processing for the Sun Performance Library routines, use one of the parallelization options (`-xparallel`, `-xexplicitpar`, or `-xautopar`) at link time, as shown in the following examples.

```
% cc  -dalign -xarch=... -xparallel a.c   -xlic_lib=sunperf
```

or

```
% f95 -dalign -xarch=... -xparallel a.f95 -xlic_lib=sunperf
```

Run-Time Issues

At run time, if running with compiler parallelization, Sun Performance Library uses the same pool of threads that the compiler does. The per-thread stack size must be set to at least 4 Mbytes with the `STACKSIZE` environment variable, as follows:

```
% setenv STACKSIZE 4000
```

Setting the `STACKSIZE` environment variable is not required for programs running with POSIX or Solaris threads. In this case, user created threads that call Sun Performance Library routines must have a stack size of at least 4 Mbytes. Failure to supply an adequate stack size for the Sun Performance Library routines might result in stack overflow problems. Symptoms of stack overflow problems include runtime failures that could be difficult to diagnose. For more information on setting the stack size of user created threads, see the `pthread_create`(3THR), `pthread_attr_init`(3THR) and `pthread_attr_setstacksize`(3THR) man pages for POSIX threads or the `thr_create`(3THR) for Sol

Degree of Parallelism

Sun Performance Library will attempt to parallelize each Sun Performance Library call according to the user's parallelization model by using either explicit threads or loop-based compiler multithreading.

The number of threads Sun Performance Library routines will attempt to use is set at run time by the user with the PARALLEL environment variable. The PARALLEL environment variable can be overridden by calls to the Sun Performance Library USE_THREADS routine.

For example, if user programs with POSIX or Solaris-thread codes are linked with `-xparallel`, `-xexplicitpar`, or `-xautopar`, each Sun Performance Library call will produce PARALLEL threads. The c

- One bound thread per CPU is created
- Each thread makes a Sun Performance Library call
- PARALLEL is set to a value greater than one

For codes using compiler parallelization, Sun Performance Library routines are parallelized with loop-based compiler directives. Because nested parallelism is not supported, Sun Performance Library calls made from a parallel region will not be further parallelized.

In the following code example, none of the calls to DGEMM is parallelized, because the loop is parallelized and only one level of parallelization is supported.

```
!$<some parallelization directive>
 DO I = 1, N
   CALL DGEMM(...)
 END DO
```

The loop consists of many DGEMM instances running in parallel with one another, but each DGEMM instance uses only one thread.

In the following code example, the loop is not parallelized.

```
DO I = 1, N
    CALL DGEMM(...)
END DO
```

If the code is linked for parallelization with `-xparallel`, `-xexplicitpar`, or `-xautopar`, the individual calls to DGEMM will be parallelized. The number of threads used by each DGEMM call will be taken from

the run-time value of the environment variable PARALLEL. However, if a higher-level loop has already parallelized this region, no further parallelization would be performed.

The number of OpenMP threads can be set by a variety of means. For example, by setting the OMP_NUM_THREADS environment variable or by setting the OMP_SET_NUM_THREADS() run-time call. If botl

The degree of parallelization within a pure-OpenMP code can be set with the OMP_NUM_THREADS environment variable. The Sun Performance Library USE_THREADS() routine can also be used to set the degree of parallelism for Sun Performance Library calls, which overrides the OMP_NUM_THREADS value.

In the following code example, each DGEMM call would be parallelized.

```
!$PAR DOSERIAL*
 DO I = 1, N
   CALL DGEMM(...)
 END DO
```

Note that the DOSERIAL* directive suppresses parallelization, but only for the loop nest within the same subroutine and it is overridden by any other directive within that nest. The DOSERIAL* directive does not impact parallelization within Sun Performance Library.

In the following code example, there will be at most two-way parallelism, regardless of the setting of the number of OpenMP threads.

```
!$OMP PARALLEL SECTIONS
 !$OMP SECTION
 DO I = 1, N / 2
   CALL DGEMM(...)
 END DO
 !$OMP SECTION
 DO I = N / 2 + 1, N
   CALL DGEMM(...)
 END DO
 !$OMP END PARALLEL SECTIONS
```

Only one level of parallelism exists, which are the two sections. Further parallelism within a DGEMM() call is suppressed.

Synchronization Mechanisms

The underlying parallelization model determines the Sun Performance Library behavior.

The two basic modes of multithreading, compiler parallelization and POSIX or Solaris threads, use two different types of synchronization mechanisms. Compiler parallelized code uses spin waits, which produce the most responsive synchronization operations, but aggressively consume CPU cycles. Compiler parallelized code produces optimal performance when each thread has a dedicated CPU, but wastes resources when other jobs or threads are also competing for CPUs.

However, codes that explicitly use POSIX or Solaris threads use synchronization functions from `libthread`. These synchronization functions are less responsive, but they relinquish the CPU when the tl

With compiler parallelization, the environment variable `SUNW_MP_THR_IDLE` can be used at run time to alter the spin-wait characteristics of the threads. Legal settings of `SUNW_MP_THR_IDLE` are as follows.

```
% setenv SUNW_MP_THR_IDLE spin

% setenv SUNW_MP_THR_IDLE 2s

% setenv SUNW_MP_THR_IDLE 100ms
```

These settings would cause threads to spin wait (default behavior), spin for 2 seconds before sleeping, or spin for 100 milliseconds before sleeping, respectively.

The link-time option `-xlic_lib=sunperf` links in Sun Performance Library functions that employ the same parallelization model as the user code, as indicated by the `-xparallel`, `-xexplicitpar`, or `-xautopar` compiler-parallelization option. Using Sun Performance Library routines does not change the s

Parallel Processing Examples

The following sections demonstrate using the `PARALLEL` environment variable and the compile and linking options for creating code that supports using:

- A single processor
- Multiple processors

Using a Single Processor

To use a single processor:

1. **Call one or more of the routines.**
2. **Link with `-xlic_lib=sunperf` specified at the end of the command line.**

Do not compile or link with `-xparallel`, `-xexplicitpar`, or `-xautopar`.

3. **Make sure the `PARALLEL` environment variable is unset or set equal to 1.**

The following example shows how to compile and link with `libsunperf.so`.

```
cc -dalign -xarch=... any.c -xlic_lib=sunperf
```

or

```
f95 -dalign -xarch=... any.f95 -xlic_lib=sunperf
```

Using Multiple Processors

To compile for multiple processors:

- Use the same parallelization option for the compiling and linking commands.
- Specify the number of processors at runtime with the `PARALLEL` environment variable before running the executable.

For example, to use 24 processors, type the following commands.

```
my_system% f95 -dalign -xparallel my_app.f -xlic_lib=sunperf
my_system% setenv PARALLEL 24
my_system% ./a.out
```

The previous example allows Sun Performance Library routines to run in parallel, but no part of the user code `my_app.f` will run in parallel. For the compiler to attempt to parallelize `my_app.f`, either `-xparallel` or `-explicitpar` is required on the compile line.

Note - Parallel processing options require using either the `-dalign` command-line option or establishi

---

To use multiple processors:

1. **Call one or more of the routines.**
2. **Link with `-xlic_lib=sunperf` specified at the end of the command line.**

Compile and link with `-xparallel`, `-xexplicitpar`, or `-xautopar`.

3. Set `PARALLEL` to the number of available processors.

The following example shows how to compile and link with `libsunperf` to enable parallel operation on multiple processor systems.

```
cc -dalign -xarch=... -xparallel any.c -xlic_lib=sunperf
```

or

```
f95 -dalign -xarch=... -xparallel any.f95 -xlic_lib=sunperf
```